UNIVERSITÀ DI PISA

Università degli studi di Pisa

Facoltà di Ingegneria

Corso di laurea in

Ingegneria Informatica

Realizzazione del supporto per il file system in un nucleo multiprogramato

RELATORI

Ing. Giuseppe Lettieri

Ing. Giuseppe Lettieri

Anno Accademico 20010/2011

Indice

Elenco delle figure

Capitolo 1

Introduzione al problema

In questo capitolo facciamo una introduzione generale al concetto di file system, mettendo in evidenza gli scopi e le varie tipologie esistenti e un introduzione al file system FAT usato su questo sistema.

1.1 Introduzione concetto file System

Definizione:

> *"Un file system è l'insieme dei tipi di dati astratti necessari per la memorizzazione, l'organizzazione gerarchica, la manipolazione, la navigazione, l'accesso e la lettura dei dati."* [**?**].

Il file system è la componente del sistema operativo che si occupa dell'archiviazione dei dati sui dispositivi di memoria secondaria e terziaria, fornendo all'utente finale un interfaccia semplice da usare realizzata mediante il concetto di *File*.

Definizione:

> *"I File è un insieme di informazioni, correlate e registrate in memoria secondaria, cui è stato assegnato un nome."* [**?**].

Il file system si occupa dell'archiviazione dei file sulla memoria secondaria, gestisce le modalità di accesso a questi e si occupa anche della protezione dei file assocciando a essi dei privileggi.Tutto questo deve essere fato in maniera trasparente rispetto all'utente finale che non deve avere nessuna conoscoenza della posizione dei dati nel disco e ne come i dischi funzionino.

1.1.1 Struttura generale

Per poter realiazzare tutte queste funzionalità il file system è composto da molti livelli distinti. La struttura illustrata alla figura 1 è un esempio di

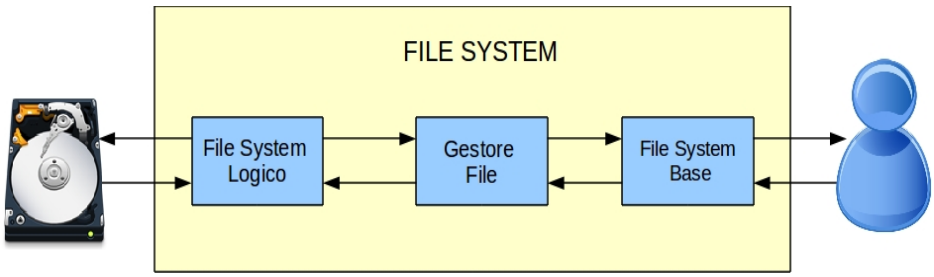
struttura stratificata . Ogni livello si server delle funzioni del livello inferiore per crearne di nuove per il livello superiore.

File system base Il File system di base è il livello più basso del file system, questo modulo ha lo scoppo di colloquiare con il driver del disco inviando dei generici comandi di scrittura e lettura.

Il modulo di organizzazione dei file Il modulo di organizzazione dei file è a conoscenza dei file e dei loro blocchi logici, cosi come dei blocchi fisici dei dischi. Conoscendo il tipo di allocazione dei file usato e la locazione dei file, puo tradurere gli indirizzi dei blocchi logici, che il file system di base deve trasferire negli indirizzi dei blocchi fisici. Il modulo di organizzazione dei file comprende anche il gestore dello spazio libero, che registra i blocchi non assegnati e li mette a disposizione del modulo di organizzazione dei file quando richiesto.

File system logico Il file system logico gestisce i metadati; si tratta di tutte le strutture del file system, eccetto gli effettivi dati (il contenuto dei file). Il file system logico gestisce le struttura della directory per fornire al modulo di organizzazione dei file le informazioni di cui necessità, dato un nome simbolico di file. Mantiene le strutture dui file tramite i blocchi di controllo dei file (file controlol block,FCB), contenti informazioni sui file, come la proprietà,i permessi e la posizione del contenuto.

Nei file system stratificati la duplicazione del codice è ridotta al minnimo. Il controllo dell'IO e, talvolta, il codice di base del file system, possono essere comuni a numerosi file system, che poi gestiscono il file system logico e i moduli per l'organizzazione dei file secondo le proprie esigenze. Sfortunatamente, la stritificazione può comportare un maggiorn overhead del sistema operativo, che può generare un conseguente decadimento delle prestazioni. L'utilizzo della strtificazione e la scelta dei numero di stratti utilizati sono aspetti critici nella realizzazione di un file system, in quanto queste scelte incidono sulle prestazioni del sistema.



1.1.2 Strutture Dati

Per realizzare un file system si usano numerose strutture dati, sia nei dischi sia in memoria. Nei dischi, il file system tiene informazioni su come eseguire l'avviamento di un sistema operativo memorizato nei dischi stessi, il numero totale di blocchi, il numero di locazioni dei blocchi liberi, la struttura delle directory e i singoli file. In questo paragrafo menzionimao le più importanti :

Blocco di controllo dell'avviamento Il blocco di controllo di controllo dell'avviamento (*b*oot control block, contiene le informazioni neccessarie al sistema per l'avviamento di un sistema operativo da quel volume; se iil disco non contiene un istema operativo, tale blocco è vuoto. Di solito è il primo blocco di un volume.

Blocchi di controllo dei volumi Il blocco di controllo dei volumi (*v*olume control block) ciascuno di loro contiene i dettagli riguradanti il relativo volume (o partizione), come il numero e la dimensiopne dei bloccho nel disco, il contatore dei blocchi liberi e i relativi puntatori, il contatore dei blocchi di controllo dei file liberi e i relativi puntatori.

Strutture delle directory La Struttura delle directory è la struttura che consente un organizzazione dei file secondo degli schemi logici, unica per ogni volume.

Blocchi di controllo di file I blocchi di controllo di file *(*FCB), contiene molti dettagli dei file, compresi i permessi di accesso ai relativi file, i prorpietari, le dimensionui e la locazione dei blocchi di dati.

Le informazioni Tenute in memoria servono sia la gestione del file sytem sia per migliorare le prestazioni attraveso l'uso di cache. Questi dati vengonon caricati in memoria quanto il volume è montato ed eliminati nel momento nel quale viene smontato.

Tabella di montaggio La tabella di montaggio interna alla memoria che contiene tuttele informazioni attinenti al ciscun volume montato

Struttura delle Directory Una parte della struttura delle directori presente nel disco viene carica in memoria in base a quali duirectory i processi hanno avuto accesso di recente.

Tabella generale file aperti La tabella generale dei file aperti contiene una coppia del FCB per ciascun file aperto all'interno del sistema operativo.

Tabella locale file aperti La tabella locale dei file aperti è una struttura dati assocciata ad ogni processo che contiene tutte le informazioni attinenti hai file aperti da quel determinato processo, tipicamente mantiene dei puntatori ai FCB della tabella globale dei file aperti.

1.1.3 Metodi di allocazione

Il problema principale che il file system deve risolvere è l'allocazione dello spazio per i file nel disco. Questo è un problema complesso da cui dipende l'efficienza del file siystem stesso, perchè un file system efficiente deve fornire una gestione dello spazio ottimale riducendo al minimo gli overhead e un accesso rapido alò contenuto dei file. Esistono tre metodi principali per l'allocazione dei file, ognuno con i suoi vantaggi e svantaggi,in base allae esigneze che si hanno e il contesto nel quale il file system andrà a lavorare si addotterà il metodo più idoneo. INTRODURRE CONCETTO DI9 FRAMENTAZIONE E LETTUREA

Allocazione contigua La base di questo metodo di allocazione consiste nell'occupare per ogni file un insieme di blocchi contigui del disco. Il principale vantaggio di questo metodo di allocazione è che si rendono trascurabili il numero di posizionamenti richiesti per accedere al contenuto del file , cosi come è trascurabile il tempo di ricerca quando quest'ultimo è neccessario. Il primo problema di cui è affetto questo metodo è l'individuazione dello spazio libero in quanto il gestore dello spazio libero deve trovare uno spazio libero formato da blocchi contigui che sia in grado di contenere la quantita di dati da immagazinare, per fare questo si possono usare degli algoritmi per la scelta del buco libero (esempi sono best-fit las-fit first-fit), ma tutti questi sono affetti dal problema della fgramenteazione esterna. Il secondo problema che si presenta usando questo metodo è che può risultare impossibile estendere il file dopo il suo allocamento, in quanto lo spazio oltre le due estremita del file pò essere gia in uso, quindi inon è possibile ampliare il file in modo contentiniuo. Questa metodologia puo essere affiancata da varie estesnsioni per renderla in grado di affrontare i problemi precedenti. Questo modello di allocazione è molto efficiente come tempi di accesso ai dati, ma le problematiche attinenti alla framentazione dello spazio lo rendono usabile solo in particolari condizioni, e anche con l'introduzione di varie estensioni questi problemi vengono ridotti ma non eliminati.

Allocazione Concatenata Il metodo di allocazione Concatenata risolve tutti i problemi di cui era afflitto il metodo di allocazione contigua.Con questo tipo di allocazione, infatti , ogni file è composto da un alista concatenata dio blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco stesso. Per poter risalire a un file nel disco la directory deve contenere un puntatore al primo blocco, e ogni blocco contiene il puntatore al blocco successivo fino all'ultimo blocco della lista, che conterra un puntaore specciale che indichera che questo è l'ultimo blocco. Quindi per eguire la lettura si legge l'indirizzo del primo blocco dalla directory fino a ragiungere la posizione ricercata

mediante la lista di blocchi. Il primo problema di questo metodo è che i file possono essere usati in modo efficiente solo con accessi sequenziali, mentre l'accesso diretto al file estremamente ineficinete in quanto bisogna scorrere tutta la lista di blocchi prima di arrivare alla posizione ricercata. Un altro svantaggio è lo spazio richiesto per il salvataggio dei puntaori dei vari blocchi. La soluzione più comune a questo problema consiste nel riunire un certo numero di blocchi contigui in **cluster**, e nell'allocare cluster invece di blocchi. Questa soluzione pero incrementa la framentazione interna che era minore con l'allocazione di blocchi. Il secondo problema riguarda l'affidabilità. Essendo i file gestiti mediante liste se si corrompe un puntatore si ha la perdita del file in quanto non è più possibili seguire la lista. Esistono varie soluzioni a questo problema, la prima consiste nell'usare liste doppiamente concatenate, oppure ussare una gestione ridondante dei puntatori. Un importante versione del metodo di allocazione Concatenata e quello **FAT** che verrà analizato in detaglio nel paragrafo §**??**, questo risolve in modo efficiente il problema dell'accesso in modo diretto.

Allocazione indicizzata Il metodo di allocazione indicizzata risolve il problema, raggruppando i putatori in una sola locazione: il **blocco indice**. Ogni file ha il proprio blocco indice : si tratta di un array d'inidirizzi di blocchi del disco. L'i-esimo elemento del blocco indice punta all'i-esimo blocco del file. Per poter risalire al file la directory contiene il puntatore al blocco indice, in questo sono efficienti sia gli accessi sequenzialie che gli accessi diretti e in più non si ha framentazione esterna. Anche questo metodo di allocazione è afflito da una problematica, che consiste nello sceglie la grandezza del blocco indice, infatti se si sceglie una grandezza troppo picola non si possono allocare file di grandi dimensioni, invece se si scegli una grandezza del blocco indice troppo elevata si introduce uno sprecco di spazio perchè la maggior parte delle dei puntatori interni del blocco indice rimangono inutilizzati. Per gestire questa situazione si usano vari approcci :

- **Schema concatenato** Un blocco indice e formato da un solo blocco del disco, e per realizzare più blocchi vengono concatenati più blocchi indice tra loro.

- **Indice a più livelli** Un blocco indice di primo livello punta ai blocchi indice di secondo livello che, a loro volta, puntano ai blocchi dei file, cosi si crea una struttura ad algbero che permette l'allocazione di grandi file.

- **Schema combinato** Consiste nell'utilizzare entrambe le soluzioni precedenti, associando ad ogni file dei puntatori diretti ai blocchi dei file , e dei puntatori a blocchi indice di vario livello in

base alle esigenze. In questo modo con gli indirizzamenti diretti si possono gestire i file di piccole dimensioni senza introdurre rilevanti perdite di spazio, e quando lo spazio indirizzato da questi risulta insufficiente si utilizzano i blocchi indice a più livelli.

Come esposto esistono una varietà di metodi di allocazione dei dati sul disco, ognuno di questi ha i suoi vantaggi e svantaggi. Il metodo di allocazione da adottare in un file system è legato al contesto in cui andrà a lavorare quel file system, soprattutto con quale tipologia di file, così facendo si cerca di ottimizzare al meglio le prestazioni. Spesso però questo non è sufficiente e per ottenere dei file system efficienti, bisogna inserire delle estensioni per migliorarne le prestazioni alle volte a discapito della generalità del file system steso.

1.1.4 Metodi di gestione dello spazio libero

Un altro aspetto importante del file system è la gestione dello spazio libero. Il sistema deve conservare una lista dello spazio libero dove sono registrate tutte le locazioni libere del disco. Per creare un file occorre cercare nella lista dello spazio libero la quantità di spazio neccessaria e assegnarla al nuovo file, quindi rimuovere questo spazio dalla lista. Quando si cancella un file, si aggiungono alla lista dello spazio libero i blocchi del disco ad esso assegnati. La lista dello spazio libero può essere implementata in vari modi:

- **Mappa di BIT** Con queo metodo si realizza la lista mediante *una mappa di bit.*Ogni blocco è raopresentato da un bit: se il blocco è libero, ilbit è 1, se il bit è assegnato il bit è 0. Il vantaggio principale è che il metodo è molto semplice da realizzare ed efficiente.

- **Lista concatenata** Con questo metodo si collegano tutti i blocchi liberi, e si tiene un puntatore al primo blocco libero in una particolare posizione del disco. Quando si ha necessità di un blocco si preleva semplicemente il primo della lista.Questo metodo non è efficiente se si ha neccessità di scorrere la lista dei blocchi.

- **Conteggio** Questo metodo, sfrutta l'idea che più blocchi liberi siano contigui tra di loro, in questo modo si può tenere traccia dell'indirizzo del primo blocco ed il numero di blocchi liberi contigui che seguono il primo blocco, e anche se la coppia indirizzo più contatore occupano piu di un indirizzo normale, la lista finale comunque ha una lunghezza inferiore.

Come nel caso della scelta del metodo di allocazione anche qui bisogna scegliere l'implementazione in base al contesto in cui ci andrà a lavorare.

1.1.5 Conclusioni

Come si è visto in questa sezione il file system è un oggetto complesso di fondamentale importanza per il sistema operativo. Il processo di realizzazione di un file system è molto lungo è complesso, e deve essere curato nei minimi particolari per cercare di ridurre al minimo gli errori, si pensi alle conseguenze che avrebbero degli errori di programmazione su una parte cosi delicata del sistema, si potrebbero avere perdite di dati, accessi non autorizzati ai file e cosi via. Nella scelta del file system da implementare abbiamo scelto il FAT per la sua semplicità implementativa tralasciando le prestazioni in quanto il contesto in cui lavorerà è un semplice sistema didattico.

Capitolo 2

FAT 32

In questo capitolo inizialmente introduciamo alcuni concetti generali riguardanti il file system FAT. Successivamente procederemmo con un maggior dettaglio analizzando le varie strutture di cui è composto e le procedure mediante le quali i dati vengono salvati e prelevati dal disco.

2.1 Carateritiche

La File Allocation Table, in sigla FAT, è un file system sviluppato inizialmente da Bill Gates e Marc McDonald. È il file system primario per diversi sistemi operativi DOS e Microsoft Windows fino alla versione Windows ME. La FAT è relativamente semplice ed è supportata da moltissimi sistemi operativi. Queste caratteristiche la rendono adatta ad esempio per i Floppy Disk e le Memory Card. Può anche essere utilizzata per condividere dati tra due sistemi operativi diversi.
Il più grande problema del File System FAT è la **frammentazione**. Quando i file vengono eliminati, creati o spostati, le loro varie parti si disperdono sull'unità, rallentandone progressivamente la lettura e la scrittura. Una soluzione a questo inconveniente è la deframmentazione, un processo che riordina i file sull'unità. Questa può durare anche diverse ore e deve essere eseguita regolarmente per mantenere le prestazioni dell'unità.
Esistono varie versioni di questo file system, in questa sede analiziamo soltanto il FAT32.
Il FAT32 presenta alcuni limiti, si possono gestire volumi che hanno una grandezza compresa tra i 512MB e 32 GB, questi limiti sono imposti dalle routine Microsoft, infatti esistono routine di terze parti che permettono di superare questi limiti.
Il file system FAT possiede anche un limite sulla grandezza massima dei file, che non può eccedere i 4GB.

2.2 Struttura

Il File system FAT usa un metodo di allocazione concatenato basato sull'utilizzo di una tabella chiamata FAT mediante la quale si possono trovare le posizioni libere dei vari cluster e rintracciare la catena dei file in modo molto semplice. La struttura generale di un volume FAT32 è composta da tre regioni principali:

1. **Regione riservata** Regione nella quale troviamo tutte le informazioni di baso livello attinenti al volume
2. **Regione FAT** Regione nella quale è presente la tabella FAT indispensabile per la allocazione dei dati sul disco.
3. **Regione Data** Regione nella quale sono presenti i dati veri e propri.

Figura 2.1: Struttura Volume FAT

2.2.1 Regione Riservata

La regione riservata è la prima regione che troviamo nel volume e contiene tutte le informazioni di basso livello necessarie per un corretto funzionamento del file system stesso. A sua volta questa regione è suddivisa in zone diverse.

- **Boot Sector** Questa è la prima struttura presente in un volume FAT, occupa il primo settore del volume (settore zero) ed è la più importante della regione riservata.
  Il Boot sector include un area chimata **BIOS Parameter Block (BPB)** che contiene alcune informazioni base per il funzionamento del file system quali ad esempio tipo del file system e posizione delle altre regioni nel disco. Il BPB occupa i byte dall'offset 0x1B fino all'offset 0x34.
  Oltre al BPB nel boot sector troviamo anche il codice del boot loader necessario per avviare un sistema operativo. La successiva tabella rappresenta schematicamente la struttura del boot sector di un volume FAT32 :

Offset	Lunghezza	Descrizione
0x00	3	Salta istruzione
0x03	8	Nome
0x0B	2	Estensione
0x0D	1	Bytes per settore
0x0E	2	Numero dei settori di riserva
0x10	1	Numero di tabelle FAT
0x11	2	FAT32 vale 0
0x13	2	Settori totali.
0x15	1	Tipo di descrittore.
0x16	2	Settori per FAT (per FAT12/16)
0x18	2	Settori per traccia
0x1A	2	Numero di testine
0x1C	4	Settori nascosti
0x20	4	Totale settori.
0x24	4	Settori occupati da una FAT
0x28	2	FAT flags
0x2A	2	Versione
0x2C	4	Cluster Root Directory
0x30	2	Numero settore FS Information
0x32	2	Numero settore di Backup
0x34	12	Riservato
0x40	1	Numero Driver
0x41	1	Riservato
0x42	1	Firma estesa
0x43	4	ID
0x47	11	Volume Label
0x52	8	FAT File sytem type
0x5A	420	Codice Boot loader
0x1FE	2	Firma boot sector

- **Fat Information Sector** Introdotto con la FAT32 per accelerare i tempi di accesso di alcune operazioni (i.e. la quantità di spazio libero), occupa generalmente il settore 1, nel record di avvio 0x30. Ha dimensione pari a 512 byte. Contiene un campo nel quale è specificato lo spazio libero rimanente (offset 0x1E8) e il primo cluster dal quale iniziare la ricerca del cluster libero (oofset 0x1F0) . La tabella successiva riepiloga i vari campi :

Offset	Lunghezza	Descrizione
0x00	4	Firma File System Info
0x04	480	Riservati
0x1E4	4	Firma File System Info
0x1E8	4	Numero cluster liberi
0x1EC	4	Cluster dal quale iniziare la ricerca
0x1F0	14	Riservato
0x1FE	2	Firma File System Info

- **Settori Riservati** Alcuni settori riservati, sono a disposizione per essere usati per rimpiazzare i settori che sono danneggiati nel disco.

2.2.2 Regione Fat

La regione FAT è la regione del disco nella quale sono salvate la tabelle FAT. Per avere una maggior sicurezza nel salvataggio dei dati sono presenti per default due tabelle FAT. Il numero di tabelle presenti nel disco e contenuto nel boot sector.

2.2.3 Regione Dati

La regione Dati e la regione che inizia successivamente alla regione FAT ed è quella nelle quali sono presenti i veri propri dati. Questa regione è vista dal file system come suddivisa in un certo numero di blocchi di grandezza uguale. Anche in questo caso tutte le informazioni necessarie per la gestione dei blocchi e più in generale della zona dati sono presenti nel Boot sector.

2.3 Getione Fat

Una partizione è suddivisa in **cluster** di egual grandezza. I cluster sono dei blocchi di spazio continuo, la cui grandezza dipende dal tipo di file system e dalla grandezza della partizione, tipicamente la grandezza di un cluster oscilla tra i 2kB e 32 kb. Ogni file puo occupare uno o piu cluster in base alla sua grandezza , quindi il file viene rapresentato da una catena di cluster (le catene sono realizate mediante una lista concatenata semplice). L'insieme di questi cluster non neccessariamente vengono salvati in posizioni adiacentin (che renderebbe piu efficiente il metodo le operaioni di lettura e scrittura) ,che quindi comporta una dispersione dei cluster per tutto il disco , questo fenomeno prende il nome di framentazione. La FAT in sé mantiene la traccia delle aree del disco disponibili e di quelle già usate dai file e dalle directory La File Allocation Table (FAT) è una lista di entry che mappano per ogni cluster della partizione. Ogni entri puo assumere i seguenti valori :

- Il numero del cluster successivo.

- Il valore speciale di fince catena (EOC)
- Il valore speciale che segnala il cluster come riservato
- Il valore speciale che segnala il cluster come malfunzionante
- Zero che identifica il cluster come libero

La struttura della fat è molto semplice, e di conseguenza anche le operazioni per la gestione dei file risultano essere semplificate. Infatti per ricercare un cluster libero si scorrere la tabella finché non si trova un cluster marcato come libero. Per rintracciare tutti i cluster di cui è formato un file è sufficiente la catena dei cluster salvata nella FAT partendo dal primo, andando a prelevare il valore contenuto nell'entry della tabella e riusarlo come indice per la successiva lettura, finché non si ottiene il valore di EOC, a quel punto tutti i settori di cui è composto un file sono stati rintracciati. I primi due entri nella FAT contengono speciali valori. La prima entry contiene una copia del media Descriptor (Descrive il device , preso dal boot sector , offset 0x15). I rimanenti bit vengono settati ad 1. La seconda entry contiene il valore di *End Of Chain*, cioè il valore usato per indicare la terminazione di una catena di cluster. I due Bit piu significativi sono usati per identificare lo stato di occupato del disco e rilevare possibile errori di montaggio. Poichè le prime due entry sono riservate il primo0 cluster utile della tabella FAT, risulta essere il 2. Il cluster 2 è il primo cluster della catena che raprensenta la root directory.

FAT32	**Descrizione**
0x00000000	Cluster Libero
0x00000001	Riservato
0x00000002-0x0FFFFFEF	Cluster usabili
0x0FFFFFF0-0x0FFFFFF6	Riservati
0xFFFFFF7	Bad Cluster
0x0FFFFFF8-0xFFFFFFFF	End Of Chain

Come si puo notare dai Valori della tabella il FAT32 utilizza solamente 28 bit per indicizzare le varie entry della tabella FAT, solitamente i 4 bit più significativi sono settati a zero, ma essendo riservati il loro valore non deve essere modificato.

2.4 Gestione Directory

Una **tabella directory** è un file speciale che che raprensenta una directory (anche conosciuta come cartella).
Ogni file o directory è rapresentata da una entrata di 32 byte nella tabella.

Ogni entrata della tabella presenta vari campi rapresentati dalla tabella successiva, il campo principale e quello che contiene il numero del primo cluster, infatti granzie a questo seguendo la catena di cluster si puo risalire al contenuto del file o cartelle. Bisogna notare che prima delle entrate della tabella delle directory ci possonono essere delle “entrate fassulle “ neccessarie per la gestione dei nomi lunghe.

Offset	**Lunghezza**	**Descrizione**
0x00	8	Nome file
0x08	3	Estensione File
0x0B	1	Attributi
0x0C	1	Riservato
0x0D	1	Tempo creazione ms
0x0E	2	Tempo creazione
0x10	2	Data di creazione
0x12	2	Data Ultimo accesso
0x14	2	2 byte piu significativi indirizzo cluster
0x16	2	Tempo Ultimo Modifica
0x18	2	Data ultimo modifica
0x1A	2	2 byte meno significativi cluster
0x1C	4	Grandezza file

Come si puo notare lo spazio per il nome del file nel suo complesso è formato da 11 byte, che risultano essere troppo pochi. Per risolvere questo problema , la microsoft a indrotto delle entrante a 32 byte, che contengono una parte del nome lungo che puo arrivare ad una lunghezza massima di 255 caratteri in formato UNICODE. All’interno delle entry lunghe sono presenti degli indici grazie ai quali e possibile seguire la lista di entry lunghe, e per essere sicuro di assocciarle all’entry corta corretta è stato introdotto un campo di checksum calcolato sul nome corto.

2.5 Differenze Specifiche

Non tutte le specifiche Microsoft esposte nei paragrafi precedenti sono state rispettate, o per aumentare l’efficienza del sistema generale o per avere delle semplificazioni realizattive. La differenze con le specifiche Microsoft sono:

- Le tabelle fat di backup vengono ignorate. Ho scelto di ignorare le tabelle di backup per una questione di efficienza, altrimenti ogni scrittura sulla fat primaria comporterebbe anche delle scritture nelle FAT di backup che comprometterebbero le prestazioni. Per migliorare quest’aspetto si potrebbe inserire un meccanismo che ad intervalli regolari

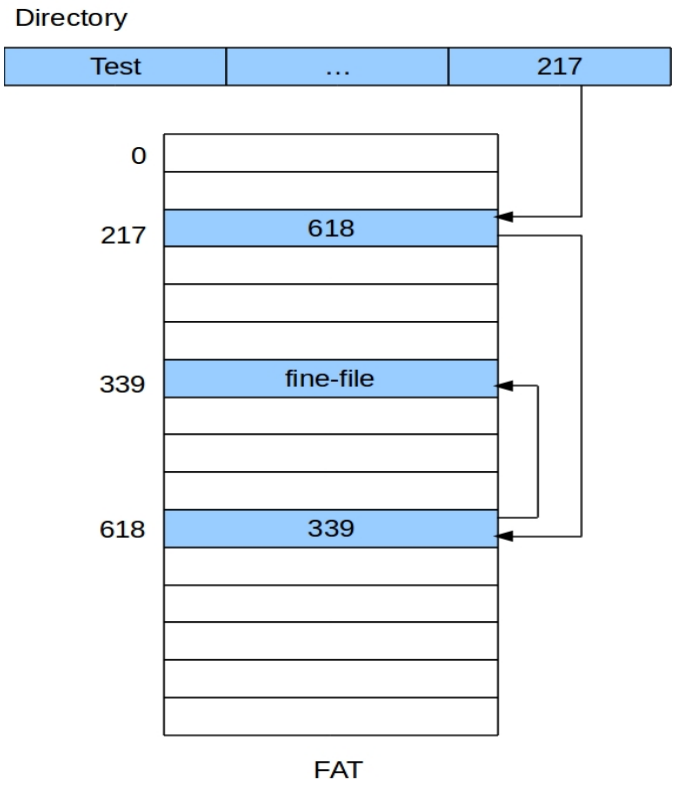


Figura 2.2: Esempio di una catena FAT

copia la FAT primaria nelle FAT di backup.

- Il parametri presenti nel settore Fat Information Sector la gestione delle zone libere del disco sono stati ignorati in quanto per semplificare e ottenere delle prestazioni migliori si è scelto di caricare l'intera FAT in memoria, quindi la ricerca di un cluster non comporta nessuna operazione sul disco.

- Ho ignorato i settori di backup del boot sector e in generale la gestione dei settori danneggiati per semplificare la gestione della FAT.

- Ho ignorato il problema della frammentazione.

2.6 Conclusione

In questo capitolo è stata fata un introduzine molto semplice al funzionamento del fat, per eventuali aprofondimenti fare riferimento alla documentazione microsoft. Come si è pottuto notare dalle spiegazioni dei capitoli precedenti il file system risulta essere molto facile da un punto di vista implentativo. Se si analizzano le prestazioni e i limiti di questo file system si nota che è poco scalbile e le prestazioni con grandi quantita di dati sono molto scarse, rendendolo quasi inutilizzabile. Un modo per migliorare le prestazioni del file ssystem è l'introduzione di un sistema di cache delle scritture / letture. Questa soluzione è quiella adottata nella maggior parte dei sistemi operativi comerciali, che grazie altra struttura modulare (spigeata nel capitolo §1) posso essere introdotti anche qui.

Capitolo 3

Analisi e specifica dei requisiti

In questo capitolo mettiamo in evidenza le esigenze di utente generico, e si mostra come il sistema risponde a queste esigenze fornendo le giuste funzionalità.

3.1 Analisi Dominio

Il nostro sistema è un file system sviluppato per il nucleo didattico del Prof. Frosini e Prof. Lettieri , più precisamente il file system verrà inserito nella versione SVN 568. Il nucleo è diviso in tre moduli distinti, sia logicamente che in fase di liking.

Modulo Sistema Il modulo sistema è il modulo nel quale si inizializza il sistema, vengono inizializzati la memoria virtuale e i processi.

Modulo IO Il modulo IO è il modulo nel quale vengono gestite le periferiche di input ed output, è in esecuzione con i privilegi Sistema

Modulo Utente IL modulo utenteè il modulo che gestisce i programmi utente, questo modulo va in esecuzione con i privilegi utente

Il file system è stato inserito nel modulo di IO perché è un componente del nucleo che necessità di usufruire di particolari istruzioni o strutture dati accessibili solo da livello sistema, necessita di poter sfruttare tutte le potenzialità della memoria virtuale necessaria per maneggiare le grandi strutture dati di cui è composto e infine sono necessarie alcune primitive bloccanti non usabili a livello sistema.

3.2 Requisiti utente

In questa sezione analizziamo i requisti utente, cioè quello che l'utente si aspetta dal sistema. L'applicazione deve fornire all'utente finale un file system funzionante compatile con le specifiche del FAT32. Il file system deve mettere a disposizione delle funzionalità per la gestione dei file e delle cartelle da parte dell'utente.
Più precisamente le specifiche dei requisiti per la gestione dei file sono :

1. L'utente deve poter creare ed eliminare i file.
2. L'utente deve poter scrivere e leggere su un File.

Le specifiche dei requisiti per la gestione delle cartelle sono :

1. L'utente deve poter creare ed eliminare le cartelle.
2. L'utente deve poter creare ed eliminare le cartelle.
3. L'utente deve porter navigare tra le cartelle.

3.3 Requisiti software

In questa sezione mettiamo in evidenza i requisiti software, cioè come il sistema soddisfa i requisiti utente.
Il sistema deve fornire un metodo semplice per la gestione dei file, si è scelto di usare un architettura simile a quella usata nei sistemi unix basata sul concetto di **descrittore di file**.
Definizione:

> *'Un descrittore di file (o file descriptor) è un numero intero non negativo che rappresenta un file aperto da un processo e sul quale il processo può effettuare operazioni di input/output.'* [**?**].

Un file presente sul disco viene individuato in modo univoco dal path, ma per poter accedere al suo contenuto occorre creare un canale di comunicazione con il kernel che renda possibile operare su di esso. Questo è necessario perché il file risiedendo sul disco, risulta essere accessibile solamente alle routine di livello sistema del kernel.
Il canale di comunicazione si crea aprendo il file con la system call *open* che inizializzerà tutte le strutture dati necessarie e restituirà il descrittore di file associato al file individuato dal path. Dopo aver aperto il file si avrà a disposizione il descrittore per effettuare le varie operazioni, ed una volta terminate queste il canale di comunicazione si dovrà chiudere usando la *close*. All'interno di ogni processo i file aperti sono identificati dal descrittore di file, ogni processo può aprire un numero limitato di file (configurabile) senza che si creino malfunzionamenti a condizione che nessun altro processo

stia usando quel file.
Tutta la gestione dei file avviene nel livello sistema del kernel, quindi è stato neccessario introdurre non semplici funzioni, ma system call che eseguano le richieste dell'utente effettuando anche il cambio di privilegio.
Le cartelle presenti nel sistema possono essere gestite mediante l'interfaccia dei file, ma esistono alcune operazioni per le quali risulta necessario usare opportune system call per la loro gestione. L'insieme di tutte le system call messe a disposizione dal kernel nel loro complesso formano il sistema File system.
Elenchiamo le system call per la gestione dei file:

int open(const char *pathname, int flags) System call che apre il file specificato da pathname, nella modalita specificata da flags e ritorna il descrittore di file.

int close(int fd) System call che chiude il file individuato da fd.

int read(fd, char * buf, size_t size) System call usata per leggere size byta dal file individuato da fd.

int write(fd, char * buf, size_t size) System call usata per scrivere size byte nel file individuato da fd.

off_t lseek(int fd, off_t offset, int whence) System call usata per spostare la posizione corrente del file individuato da fd.

int remove(const char *pathname) System Call che rimuove il file specificato da pathname.

int rename(const char *oldpath, const char *newpath) System call che rinomina il file identificato da oldpath con quello specificato da newpath.

Elenchiamo le system call per la gestione delle cartelle:

int mkdir(const char *dirname) System call che crea una directory.

int rmdir(const char *dirname) System call che elimina la directory individuata da dirname, solamente se questa è vuota.

int chdir(const char *path) System call che modifica la directory corrente.

char *getcwd(char *buf, size_t size) System call che riporta la directory corrente.

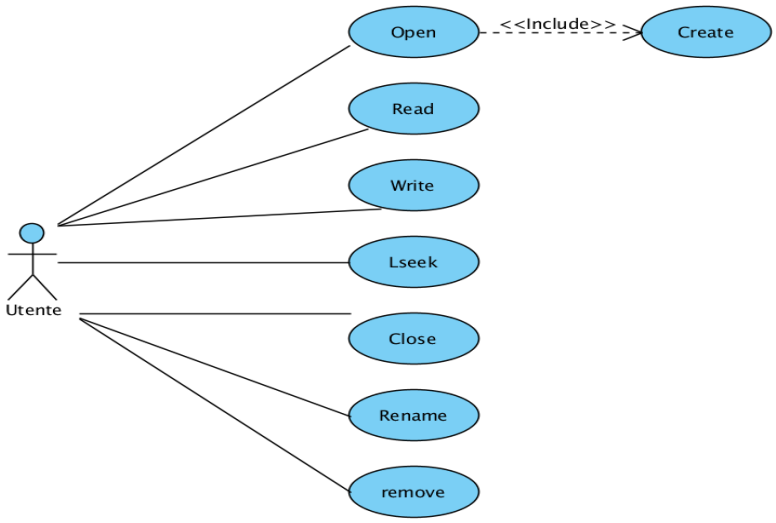


Figura 3.1: Diagramma Caso d'uso gestione File

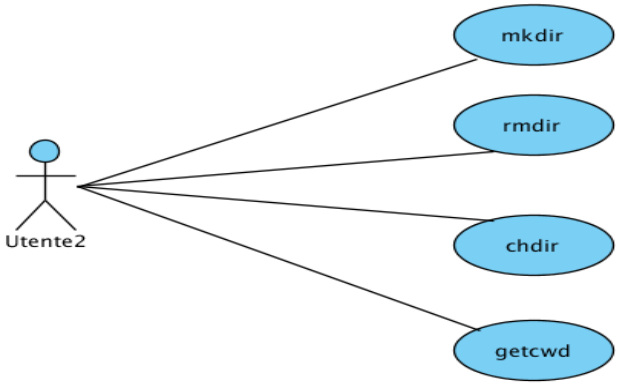


Figura 3.2: Diagramma Caso d'uso gestione Cartelle

Capitolo 4

Modifiche Introdotte sul nucleo

Il file system è un componente complesso del modulo IO, per far fronte a tutte le esigenze di questo componente è stato necessario introdurre alcune modifiche e nuove funzionalità ai moduli di sistema e IO. Senza l'introduzione di tali modifiche non sarebbe stato posssibile realizzare il supporto al file system FAT32, ma nonostante ciò queste non fanno parte del file system.

4.1 Modifiche modulo Sistema

Il modulo sistema è il modulo principale del nucleo. Questo modulo viene caricato dal boot loader ed esegue l'inizializzazione delle principali caratteristiche del nucleo, quali la gestione dei processi e la gestione della memoria virtuale e la gestione dei semafori e poi carica i restanti moduli. Le modifiche introdotte su questo modulo riguardano principalmente due aspetti, la gestione dell'hard disk e l'introduzione di una area riservata per ogni processo.

Hard Disk

Il driver dell'hard disk è in inserito nel modulo sistema, poiché è necessario per una corretta gestione dell'area di swap. Le modifiche introdotte riguardano la gestione dei volumi di un disco che era assente.Le primitive presenti supportano le operazioni di scrittura e lettura sul disco ma senza aver coscienza del volume sul quale si sta lavorando. Per un corretto funzionamento del filesystem è stato necessario introdurre delle primitive accessibili solamente a livello sistema che mi permettessero di identificare quali volumi sono presenti sul disco ed effettuare operazioni di lettura e scrittura su questi.

```
natl  p_id;            // id del processo padre
void* iostate;         // puntatore area riservata
natl  iostate_size;    // grandezza area riservata
```

get_partizione E' una primitiva a livello sistema che mi permette di accedere alla lista delle partizioni presente nella memoria del modulo sistema altrimenti inaccessibile dal modulo IO in quanto non collegati tra di loro. Questa primitiva riporta un indice univoco mediante il quel è possibile riferire la partizione all'interno del disco.

write_part_n Primitiva a livello sistema che mi permette di eseguire le operazioni di scrittura sulle varie partizioni presenti sul disco. Per poter identificare la partizione in modo univoco, associate alle informazioni attinenti il canale ATA e il DRIVER si deve indicare anche l'indice ottenuto dalla primitiva get_partizione

read_part_n Primitiva a livello sistema che mi permette di eseguire le operazioni lettura sulle varie partizioni presenti sul disco. Per poter identificare la partizione in modo univoco associate alle informazioni attinenti il canale ATA e il DRIVER si deve indicare anche l'indice ottenuto dalla primitiva get_partizione

Area privata processo

Il file system per poter gestire correttamente i file e le directory ha la necessità di associare delle informazioni private ad ogni processo. Il nucleo non supportava questa possibilità, quindi è stato necessario introdurre un meccanismo che mi permettesse di gestire quest'area privata dallo spazio IO.
La prima modifica introdotta riguarda il descrittore di processo nel quale sono stati introdotti dei campi aggiuntivi che mi permettessero di associare e gestire l'area privata di ogni processo. La prima questione da affrontare, una volta inserite queste modifiche è come e con quali valori l'area deve essere inizializzata. L'area privata di un processo viene inizializzata durante la creazione del processo copiando i valori contenuti nell'area del processo padre, se questa è presente. Se l'area del processo padre non è presente, si copiano i valori di default contenuti nell'area di memoria globale inserita nel modulo sistema ed individuata dal puntatore *io_global_info*, se è stata inizializzata. Se non è presente nessun area globale, l'area privata del processo non viene inizializzata. Per inizializzare in fase di creazione ho modificato la crea_processo nel seguente modo.

Questa area però deve anche essere dealocata, questo compito viene eseguito all'interno della *distruggi_processo*.

```
        pdes_proc->p_id = esecuzione->id;
        father_proc=des_p(esecuzione->id);

        // non puo esistere la condizione nella
        // quale le informazioni globali non sono
        // inizzializatementre quelle del padre si,
        // il contrario invece puo capitare
        // ( processo main )

        if(io_global_info && io_global_size
              && father_proc->iostate &&
               father_proc->iostate_size ) {
            // copio le informazioni del padre
            natl size=father_proc->iostate_size;
            pdes_proc->iostate_size=size;
            pdes_proc->iostate=alloca(size, 1);
            if (!pdes_proc->iostate) goto errore6;
            memset(pdes_proc->iostate, 0, size);
            memcpy(pdes_proc->iostate,father_proc->iostate,size);

        }else if (io_global_info && io_global_size) {
           // se il padre non ha informazioni copio quelle globali
            pdes_proc->iostate_size=io_global_size;
            pdes_proc->iostate=alloca(io_global_size, 1);
            if (!pdes_proc->iostate) goto errore6;
            memset(pdes_proc->iostate, 0, io_global_size);
            memcpy( pdes_proc->iostate,io_global_info,io_global_size);

        } else {
          //azzero le informazioni riguardanti l'area IO
            pdes_proc->iostate=NULL;
            pdes_proc->iostate_size=0;
        }
}
```

Poiché l'area privata è stata inserita nello spazio di indirizzamento di sistema, per il file system e qualunque altro componente del modulo IO è impossibile accedere alle informazioni presenti nei descrittori di processo, e di conseguenza all'area privata.
Per gestire quest'area in modo corretto ho inserito le seguenti primitive accessibili solo da livello sistema :

c_io_space_init Primitiva che inizializza l'area globale presente nel modulo sistema. Questa primitiva alloca lo spazio necessario per inserire i valori ricevuti come parametri.

c_io_space_read Primitiva che mi permette di leggere dei valori nell'area privata di un processo, specificando un offset di interesse.

c_io_space_write Primitiva che mi permette di scrivere dei valori nell'area privata di un processo, specificando un offset di interesse.

4.2 Modifiche modulo IO

Nel modulo IO, e il modulo nel quale è inserito il file system che verrà discusso ampiamente nei prossimi capitoli, in questo paragrafo mi limito a mostrare le modifiche introdotte nel modulo IO non facenti parte del file system.
Nel modulo IO le principali modifiche che ho introdotte sono, l'introduzione di un sistema per la gestione dell'ora e della data, l'introduzione di una semplice gestore degli errori e l'introduzione di alcune librerie.

Data e ora

Il nucleo non possiede nessun meccanismo per ottenere ne l'ora corrente, ne la data corrente. Ogni computer è munito di un sistema per mantenere la data e l'ora dopo lo spegnimento del computer, questo sistema e formato da una memoria CMOS e da un RTC accessibili mediante delle porte di IO. Per usare questo dispositivo ho scritto un piccolo driver gestito a controllo di programma che mi permettesse di leggere la data e l'ora contenute nei suoi registri e operasse le opportune conversioni sui valori letti fornendomi un interfaccia semplice ed omogenea. Per usufruire di questo meccanismo ho inserito due primitive accessibili anche da livello utente :

get_time Primitiva di livello utente che riporta l'ora corrente. L'ora corrente viene restituita in un formato compatto, di grandezza pari a 4 byte, ogni byte assume un significato particolare.

- Byte 0: Contiene il valore del secondo corrente espresso in formato binario. Range [0-59].

- Byte 1: Contiene il valore del minuto espresso in formato binario. Range [0-59].
- Byte 2: Contiene il valore dell'ora corrente espresso in formato binario.Range [0-23].
- Byte 3: Byte inutilizzato a disposizione.

get_data Primitiva di livello utente che riporta la data corrente. La data corrente viene restituita in un formato compatto, di grandezza pari a 4 byte, ogni byte assume un significato particolare.

- Byte 0: Contiene il valore del giorno corrente espresso in formato binario. Range [1-31].
- Byte 1: Contiene il valore del mese corrente espresso in formato binario. Range [1-12].
- Byte 2: Contiene il valore dell'anno corrente espresso in formato binario. Range [0-99].
- Byte 3: Byte inutilizzato a disposizione.

4.3 Librerie

Data la grandezza del progetto è stato necessario implementare delle librerie più meno standard che utilizzo in quasi tutti i moduli. Le librerie introdotte vengono linkate al modulo IO, quindi sono disponibli per tutto il sistema IO.

string.h Libreria standard C, dove sono presenti le principali funzioni per la gestione delle stringhe.

strarg.h Libreria standard C, sono presenti le macro e le funzioni per la getsione degli argomenti delle funzioni.

wchar.h Libreria standard C, dove sono presenti le principali funzioni per la gestione delle stringhe in formato wchar.

error.h Libreria standard C, piccola libreria che permette di gestire in modo molto semplice gli errori a livello sistema.

4.4 Gestione errori

Sempre a causa della grandezza del progetto, è stato neccessario introddure una semplice gestione degli errori, realizata dalla libreria errno.h. Questa gestione viene realizata principlaemente mediante delle funzioni che agiscono su delle varibili globali che contengono lo stato attuale.

- **errno** : word che contiene un codice identifgicativo per l'errore che si è verificato.
- **str_errno** : stringa che contiene la spiegazione dell'errore contenuto in errno.

Queste variabili devono essere modificate mediante le seguenti funzioni, e anche se è possibile accederci diretamente conviene usare le funzioni.

perror Funzione che stampa la stringa di errore accodata alla stringa passata come argomento

set_errno Funzione che setta le variabili di errore globali ai valori specificati come argomento.

reset_errno Funzione che resetta le variabili di errore, inserendo il valore di default (successo).

4.5 Conclusioni

Capitolo 5

Struttura del progetto

In questo capitolo mostriamo l'architettura software del progetto, mettendo in evidenza i moduli di cui è composto e le varie relazioni che sussistono tra di loro.
Prima di tutto bisogna definire il concetto di modulo :

> *"Un modulo software è una porzione di software, che conteinee e fornisce servizi o risorse e a sua volta può usufruire di risorse o servizi offerti da altri moduli. L'insieme dei servizi messi a disposizione prende il nome di interfaccia."*

¡¡¡¡¡¡¡ Ogni modulo introdotto in questo capito vera ripresenso e approfondito successivamente. Si è deciso di adottare una struttura modulare per ottenere una maggior chiarezza e più facile gestione del sistema file system. Ogni modulo è composto da una o più componenti, ognuna delle quali svolge un compito ben preciso. Un ulteriore vantaggio di questo approccio è quello che qualora si volesse introdurre delle modifiche, per ottenere miglioramenti o nuove funzionalità di apportare tali modifiche solamente al modulo interessato senza che ciò comporti variazioni o malfunzionamenti sufli altri moduli. Per esempio se uno volesse introdurre una cache delle scritture, questa modifica riguarderebbe solamente il modulo a cui è stato assegnato il compito di gestire la zona dati del disco, mentre gli altri moduli non subirebbero modifiche.

5.1 Struttura Modulare

Il File system è composto da tre macro-moduli, che riprendono la struttura generale di un file sytem vista nel precedente §1 capitolo. Seguendo un approccio bottom-up descriviamo il funzionamento generale di questi moduli.

Modulo Base Il primo modulo è il modulo base.Lo scoppo di questo modulo è di interfacciarsi con il driver del disco e di fornire al modulo

superiore un interfaccia con la quale gestire le regioni FAT e Data del disco.

Modulo Intermedio Il secondo Modulo è il modulo intermedio, che esegue la gestione logica dei file. Si occupa della gestione entry FAT e dei nomi. Fornendo un interfaccia semplice al modulo superiore mediante la quale si possono gestire i file tramite il concetto di file control block.

File system logico Mette a disposizione le system call usate dall'utente finale, e traducce le informazioni ottenute dall'utente con le informazione comprese dal sistama sottostante.

/// figura

5.2 Modulo Base

Il modulo base è il modulo che si occupa della gestione a basso livello del file system. Lo scopo è quello di gestire in modo corretto le zone dati e fat del disco e di mostrare ai livelli superiori un interfaccia semplice con la quale interagire. Il compito di questo modulo rimane comunque molto complesso, quindi si è scelto di scomporre questo modulo in componeti piu semplici alle quali e stato assegnato un ben determinato scopo.

5.2.1 Volume

Il primo componente chiamato dal sistema è il modulo VOLUME. Questo ha il compito di analizzare il disco alla ricerca delle partioni fat e di creare la tabella dei volumi. La tabella dei volumi è una stuttura globale, necessaria ai moduli superiori per una corretta gestione dei volumi presenti nei dischi. E' una struttura molto semplice realizzata mediante una lista concatenata nella quale vengono inseriti tutti i moduli rilevati. Per ogni modulo è presente un semaforo per la gestione della mutua esclusione della regione data e regione fat un mpo fat info che contiene tutti i dati specifici di quella partione e le struttura neccessarie alla gestione del volume per i livelli successivi puntatore alla fat e puntatore al FCB della root. Il modulo mette a disposizione la seguente interfaccia : create tabella dei volumi che crea la tabella. La funzione per poter creare la tabella cerca nel disco mediante la primitiva di sistema get partizione le partioni fat, una volta identicifcate crea un nuovo elemento nel quale inserisce tutte le informazioni di quel volume , assegna un etichetta univoca al volume e crea ed inizliazza il fcb della root directory. print tabella dei volumi è una funzione di debug, che viene usata per stampare a video il contenuto della tabella. get_volume funzione di utilittà neccessaria per rintracciare un determinato volume mediante l'etichetta delete_tabella funzione che viene chiamata esclusivamente in caso di errore per rilasciare le varie zone dati allocate

5.3 Fat

Il Secondo componente del modulo base è il modulo fat , che ha il compito di gestire in modo corretto la tabella fat. A questo livello tutti i file vengono gestiti mediante le catene di cluster che sono salvate nella fat. Quindi il compito principale di questo modulo è quello di offrire un interfaccia semplice che permetta la creazione, la modifica e l'eliminazione di queste catene Per questioni di efficenza si è scelto di caricare la tabella fat in memoria, quindi la tebella fat deve essere sempre in uno stato consistente con quella presente nel disco, quindi le scritture devo comportare una scrittura nel disco e una in memoria e poiche si poterbbero verificare delle incosistenze tra i dati le scritture devono avvenire in mutua esclusione tra di loro, questo viene garantito dal semaforo presente nel volume assocciato a questa FAT. Tutto questo viene eseguito dalle funzioni di scrivi fat che mi garantisce che la ricerca del cluster libero e la scirttura in memoria e sul disco avengano in mutua esclusione, Ilmodulo esporta la seguente interfaccia : init_fat funzione che carica in memoriala tabella fat delete memory fat funzione elimina la tabella fata dalla memeoria solitamente usata in casi di errore create fat crea una nuova catena fat append fat aggiunge un cluster alla catena delete fat elimina l'ultimo cluster dalla catena delete_fat_all elimina tutta la catena get_next_fat serve per scorrere una catena fat, quest fuznione esegue dei controlli sulla consistenza della catena stessa print chain funzione di debug che stampa a video la catena

5.3.1 DATA

Questo componente ha lo scopo di gestire la zona dati. Questo modulo mette a disposizione degli altri moduli le funzioni di scrittura e lettura sul disco senza che questi si preoccupi di quali settori siano coinvolti nella scrittura o lettura. In questo modo si offre un visione uniforme dello spazio del disco, semplificando le operazioni di lettura e scrittura per i moduli superiori.

5.4 Modulo Intermedio

In questo modulo avviene la gestione delle entry presenti nel disco. Il modulo ijntermedio fa da intermediario tra il modulo logico che ottiene le informazioni dall'utente e il modulo base. Il modulo intermedio colloquia con il modulo logico mediante una struttura dati che prende il nome di file control block, nella quale sono presenti tutte le indormazioni per una corretta gestione dei file. Estrae le informazioni dai fcb passati dal livello superiore e esegue le richieste corrette al modulo inferiore per assolvere la richiesta. Il modulo e composto da due componenti : il modulo direntry che il modulo che si occupa della gestione delle entry presenti nel disco il modulo

fcb, e un semplice componente che racchiude le strutture dati neccessarie per la gestione dei file!

5.4.1 Logico

Questo modulo è il modulo che esporta tutte le system call che sono accessibili da livello utente . preleva le informazioni e le inserisce nei vari fcb. In questo modulo gestisce anche la regione di dati riservata nel modulo sistema neccessaria per il salvataggio delle informazioni private per ogni processo.